

NPACI All-Hands Meeting 2002
Tutorial on Math Libraries

Aztec

Osni Marques
Lawrence Berkeley National Laboratory (LBNL)
National Energy Scientific Computing Center (NERSC)
(*osni@nersc.gov*, *http://www.nersc.gov/~osni*)

Outline of the Talk

- Aztec
 - Features
 - Basic steps
 - Applications
- State-of-the-art
 - Trilinos
 - Hypre

Aztec

<http://acts.nersc.gov/aztec>

Solves iteratively large sparse linear systems of equations of the form

$$Ax = b$$

such as those arising from applications that model complex physics problems using differential equations (e.g. finite differences or finite element methods)

Aztec: solvers and preconditioners

- Implements Krylov iterative methods (CG, CGS, Bi-CG-Stab, GMRES, TFQMR)
- Suite of preconditioners (Jacobi, Gauss-Seidel, overlapping domain decomposition with sparse LU, ILU, BILU within domains)
- Highly efficient, scalable (1000 processors on the “ASCI Red” machine)

Aztec: *basic steps*

- Prepare your linear system
 - distribute the matrix
 - call **AZ_transform**
 - set up right-hand side and initial guess
 - call **AZ_reorder_vec** on initial guess and right-hand side
- Select an iterative solver and a preconditioner
- Call **AZ_solve**
- Call **AZ_invorder_vec** on solution

```

#include <stdio.h>
#include <stdlib.h>
#include "az_aztec.h"

#define perror(str) { fprintf(stderr,"%s\n",str); exit(-1); }

int n = 6;           /* POISSON EQUATION IS SOLVED ON an n x n GRID. This */
                     /* corresponds to an x-vector of length n-squared and */
                     /* thus the matrix A is of size n-squared by n-squared. */
#define MAX_NZ_ROW 5 /* Max number of nonzero elements in any matrix row */

extern void create_matrix_row(int row,int i,double val[],int bindx[]);

int main(int argc, char *argv[])
{
    /* Set up a Poisson test problem and solve it with AZTEC. */
    {
        double *b,*x;           /* rhs and approximate solution */
        int i, nrow;

        /* See Aztec User's Guide for the variables that follow */
        /* Omitted to save space */

        /* get number of processors and the name of this processor */
        MPI_Init(&argc,&argv);
        AZ_set_proc_config(proc_config, MPI_COMM_WORLD);

        /* Define partitioning: matrix rows (ascending order) owned by this node */

        nrow = n*n;
        AZ_read_update(&N_update, &update, proc_config, nrow, 1, AZ_linear);

        /*
         * Create the matrix: each processor creates only rows appearing in update[]
         * (using global col. numbers).
         */

        bindx = (int *) malloc((N_update*MAX_NZ_ROW+1)*sizeof(int));
        val = (double *) malloc((N_update*MAX_NZ_ROW+1)*sizeof(double));
        if (val == NULL) perror("Error: Not enough space to create matrix");

        bindx[0] = N_update+1;

        for (i = 0; i < N_update; i++) {
            create_matrix_row(update[i], i, val, bindx);
        }
    }
}

```

$$\frac{d^2v(x,y)}{dx^2} + \frac{d^2v(x,y)}{dy^2} = f(x,y)$$

$$\begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & & -1 & 4 \end{bmatrix}$$

main (cont.)

```
/* convert matrix to a local distributed matrix */

AZ_transform(proc_config, &external, bindx, val, update, &update_index,
            &extern_index, &data_org, N_update, NULL, NULL, NULL,
            AZ_MSR_MATRIX);

/* initialize AZTEC options */

AZ_defaults(options, params);

/* Set rhs (delta function at lower left corner) and initialize guess */

b = (double *) malloc(N_update*sizeof(double));
x = (double *) malloc((N_update + data_org[AZ_N_external])*sizeof(double));
/* NOTE: SOLUTION VECTOR MUST CONTAIN SPACE FOR EXTERNAL ELEMENTS */
if ((x == NULL) && (i != 0)) perror("Not enough space in rhs");
for (i = 0; i < N_update; i++) {
    x[update_index[i]] = 0.0;
    b[update_index[i]] = 0.0;
    if (update[i] == 0) b[update_index[i]] = 1.0;
}

/* solve the system of equations using b as the right hand side */

AZ_solve(x, b, options, params, NULL, bindx, NULL, NULL, NULL, val, data_org,
          status, proc_config);

/* Free allocated memory */

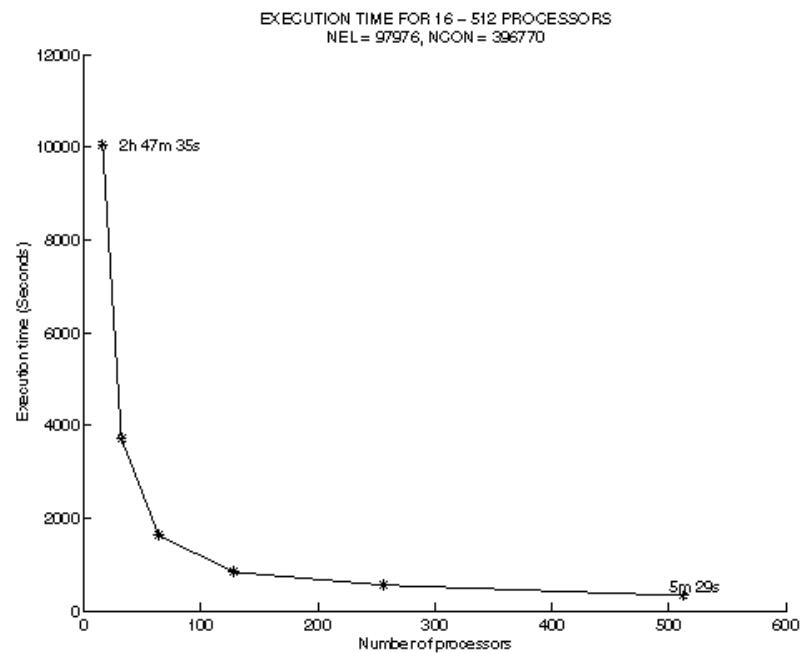
free((void *) update);    free((void *) update_index);
free((void *) external);  free((void *) extern_index);

free((void *) x);         free((void *) b);           free((void *) bindx);
free((void *) val);       free((void *) data_org);

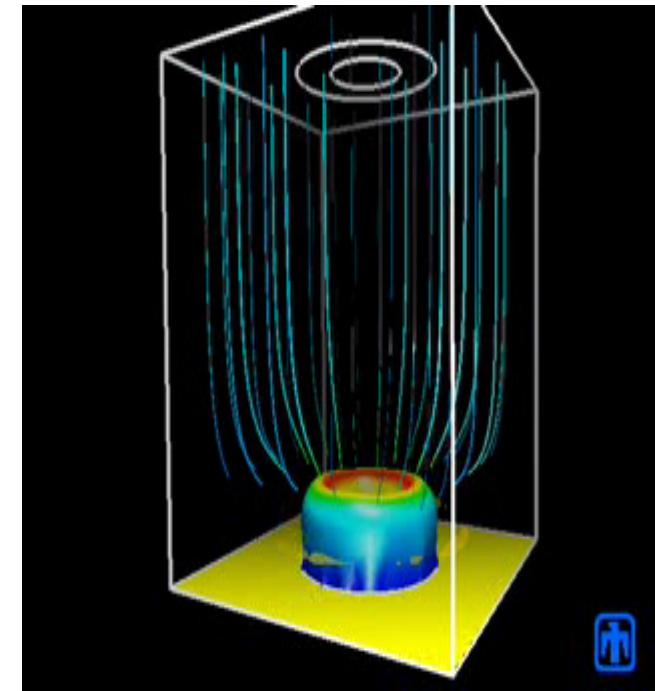
MPI_Finalize();
return(1);

}
```

Aztec: *applications*



TOUGH2 (Transport Of Unsaturated Groundwater and Heat) code, transport simulation in porous and fractured media (LBNL).



Co-flowing Annular Jet Combuster, a parallel 3D pseudo-transient simulation to steady state operation; MPSalsa code (SNL).

Trilinos

<http://www.cs.sandia.gov/Trilinos>

- Trilinos is a multifaceted solver project.
- Encompasses efforts in:
 - Linear solvers.
 - Eigensolvers.
 - Nonlinear and time-dependent solvers.
 - Others.
- Provides a common framework for current and future solver projects.
- Specifically provides:
 - A common set of concrete linear algebra objects for solver development and application interfaces.
 - A consistent set of solver interfaces via abstract classes (API) .

Trilinos Concrete Solver Components

- Linear systems:
 - Multi-level preconditioners (ML).
 - Robust algebraic preconditioners (IFPACK).
 - Complex solvers (Komplex).
 - Block iterative methods (BGMRES, BLCG).
 - Object-oriented C++ Aztec (AztecOO).
- Eigensystems:
 - Scalable generalized eigensolver (ANASAZI).
- Nonlinear systems:
 - Suite of nonlinear methods (NLS).

Trilinos: *AztecOO*

- Aztec is the workhorse solver at SNL:
 - Extracted from the MPSalsa reacting flow code.
 - Installed in dozens of SNL applications.
 - 800+ external licenses.
- AztecOO leverages the investment in Aztec:
 - Uses Aztec iterative methods and preconditioners.
- AztecOO improves on Aztec by:
 - Using objects for defining matrix and RHS.
 - Providing more preconditioners/scalings.
 - Using C++ class design to enable more sophisticated use.
- AztecOO interfaces allows:
 - Continued use of Aztec for functionality.
 - Introduction of new solver capabilities outside of Aztec.

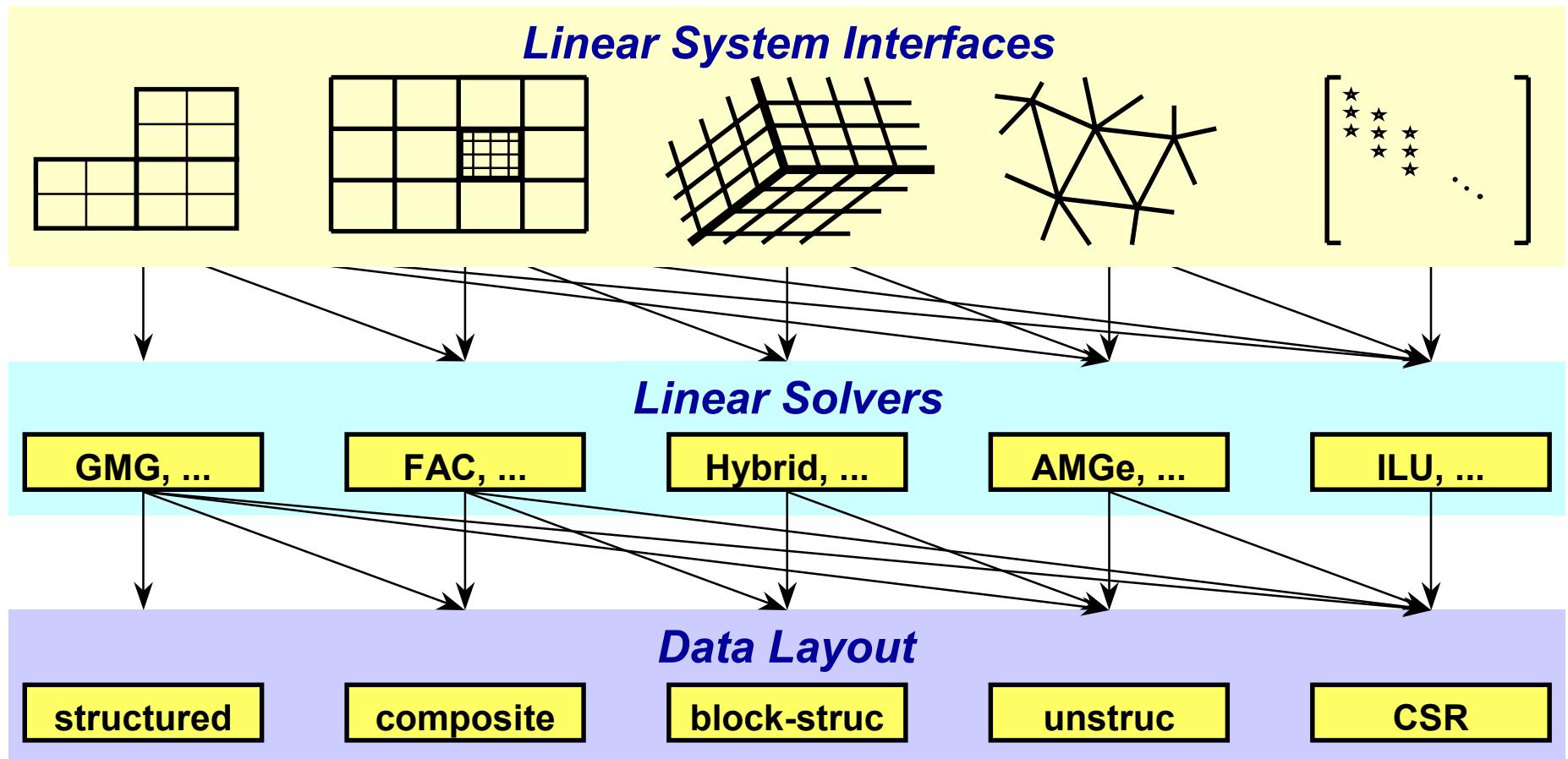
Hypre

<http://acts.nersc.gov/hypre>

- Before writing your code:
 - choose a conceptual interface
 - choose a solver / preconditioner
 - choose a matrix type that is compatible with your solver / preconditioner and conceptual interface
- Now write your code:
 - build auxiliary structures (e.g., grids, stencils)
 - build matrix/vector through conceptual interface
 - build solver/preconditioner
 - solve the system
 - get desired information from the solver

Hypre: *interfaces*

Multiple interfaces are necessary to provide “best” solvers and data layouts



Hypre: *Why multiple interfaces?*

- Provides natural “views” of the linear system
- Eases some of the coding burden for users by eliminating the need to map to rows/columns
- Provides for more efficient (scalable) linear solvers
- Provides for more effective data storage schemes and more efficient computational kernels

Hypre: *conceptual interfaces*

- Structured-Grid Interface (**Struct**)
 - applications with logically rectangular grids
- Semi-Structured-Grid Interface (**SStruct**)
 - applications with grids that are mostly structured, but with some unstructured features (e.g., block-structured, AMR, overset)
- Finite Element Interface (**FEI**)
 - unstructured-grid, finite element applications
- Linear-Algebraic Interface (**IJ**)
 - applications with sparse linear systems

Solvers	System Interfaces			
	Struct	SStruct	FEI	IJ
Jacobi	X			
SMG	X			
PFMG	X			
BoomerAMG	X	X	X	X
ParaSails	X	X	X	X
PILUT	X	X	X	X
PCG	X	X	X	X
GMRES	X	X	X	X

Hypre: *setup and use of solvers*

- Create the solver

```
HYPRE_SolverCreate(MPI_COMM_WORLD, &solver);
```

- Set parameters

```
HYPRE_SolverSetTol(solver, 1.0e-06);
```

- Prepare to solve the system

```
HYPRE_SolverSetup(solver, A, b, x);
```

- Solve the system

```
HYPRE_SolverSolve(solver, A, b, x);
```

- Get solution info out via conceptual interface

```
HYPRE_StructVectorGetValues(struct_x, index, values);
```

- Destroy the solver

```
HYPRE_SolverDestroy(solver);
```

Development Teams

- Trilinos (SNL)
 - Mike Heroux
 - Teri Barth
 - David Day
 - Rob Hoekstra
 - Rich Lehoucq
 - Kevin Long
 - Roger Pawlowski
 - Ray Tuminaro
 - Alan Williams
- Hypre (LLNL)
 - Robert Falgout
 - Andy Cleary
 - Jim Jones
 - Edmond Chow
 - Van Henson
 - Chuck Baldwin
 - Peter Brown
 - Panayot Vassilevski
 - Ulrike Meier Yang